AMD DEVELOPER symposium 2002

The Portland Group Compiler Technology

STMicroelectronics

PGI Scalar and Parallel Compilers, Debuggers and Profilers for Hammer

The Portland Group Compiler Technology

STMicroelectronics More Intelligent Solutions

Overview

- PGI History
- Compilers, Debuggers, Profilers, Tools
  - Parallel Programming
  - Fortran, C & C++ Compilers
  - Debugger & Profiler
  - Cluster Development Kit
- Schedule
- Wrapup

PGI History

- **1989** - Spin-off from Floating Point Systems (FPS Computing)
- **1990-93**
  - Vector, SW Pipelining, i860 Parallel C, C++, F77 compilers, Debugger, Profiler
  - **#1 in Performance and in Market-share (70%),** 20+ OEM vendors
- **1994-99**
  - Parallel PGHPF™ HPF compiler for most SMP and MPP systems
  - **#1 in Performance and Market-share (60%).** Cray, HP, Intel OEMs
- **1996-2000**
  - Parallel Pentium Pro C, C++, F77, Profiler for Linux, Solaris86, NT
  - Cluster Development Kit (CDK™) for Linux86
  - **#1 in Fortran and SMP Parallel Performance**
  - **#1 in Market-share of commercial Linux86 Fortran and CDK™**
- **Dec 2000** – Acquired by STMicroelectronics
  - Rebrand to **The Portland Group Compiler Technology**
  - DSP Architectures in addition to HPC
- **Jan 2001** – OpenMP & MPI Cluster Debugger & Profiler For Linux
- **Aug 2002** – **Collaboration w/ AMD for Hammer64 & Hammer32**

CDK Compiler & Tool Support

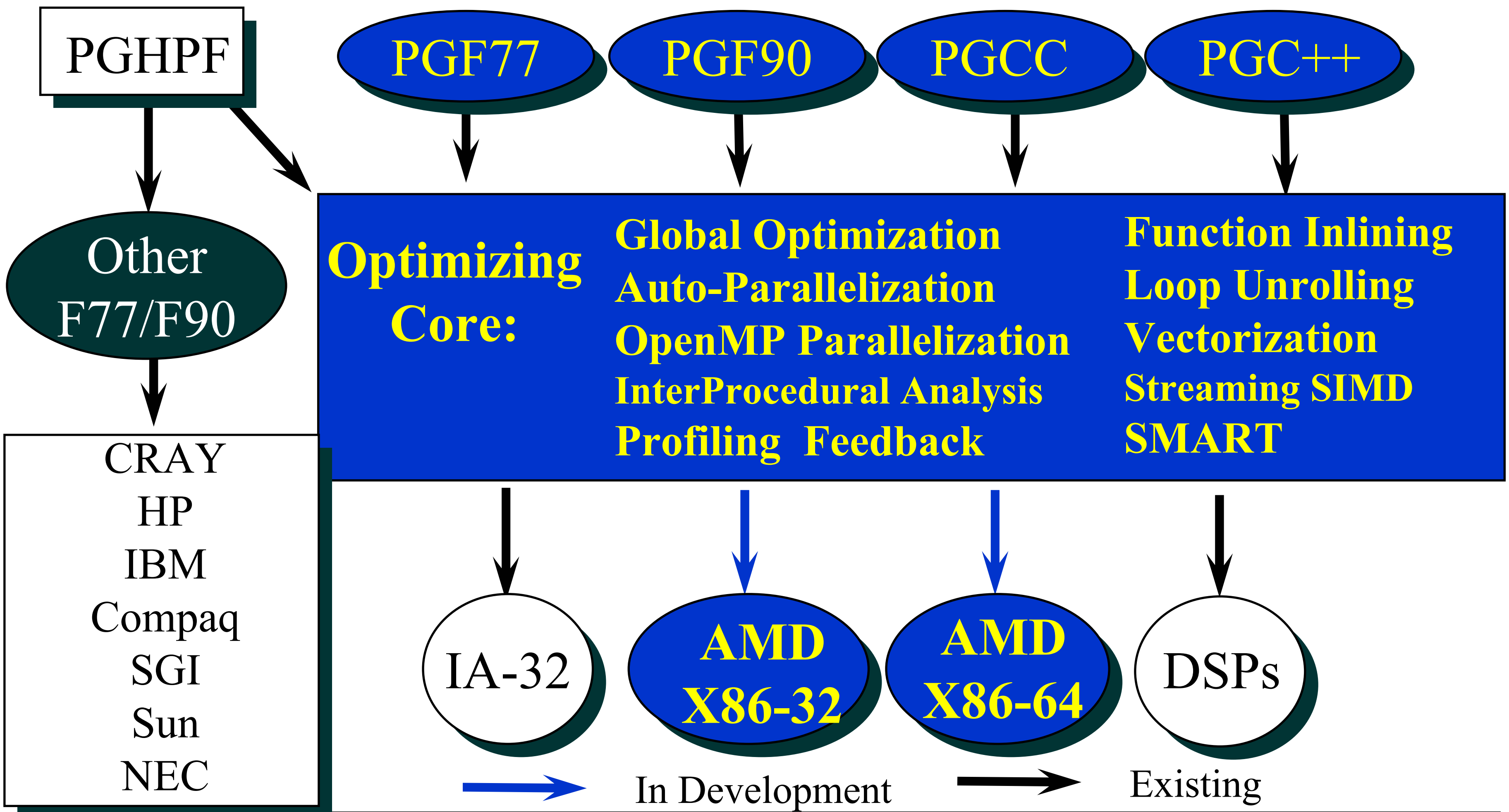
- AMD and STMicroelectronics are collaborating to bring The Portland Group Compiler Technology to x86-64
  - Cluster Development Kit® package being retargeted
  - CDK™ includes
    - F90, F77, C, C++ Scalar and OpenMP SMP parallel Compilers
      - Optimized 32-bit and 64-bit code generation
        - Focus on Fortran performance
    - Scalar & parallel PGDBG® debugger and PGPROF® post-mortem profiler
      - Multi-Threaded OpenMP, Multi-Process MPI Cluster and Hybrid Debugging and Profiling.
    - Linux and Windows, GUI
    - Turn-key installation, PBS, MPI-CH, Scalar/Parallel Math Libs, Training
  - Fortran Performance Goals
    - 32-bit code to outperform that of any other 32-bit compiler
    - 64-bit code performance within 95% of 32-bit code on same Hammer platform. Medium-term goal is to exceed 32-bit code performance.

Good News

- **The good news is that the Hammer32 and Hammer64 architectures are simply advancements from the existing 32-bit x86 efforts that currently exist.**
  - Hammer improves in a number of ways
  - Isn't this what we all wanted?
- **As a result, PGI's effort to deliver world-class compiler performance to the Hammer32 and Hammer64 architectures for Linux and Windows will be very straight-forward.**
  - Very low risk – excellent leverage of existing technologies and products.
  - More registers and better ABI provide better opportunities to use more recent compiler technologies and to enhance compiler performance.
  - Compilers and tools will still require quite a bit of work to retarget, port and tune but the end result will be worth the effort!
- **Software Development Programming Model will not be changed**
  - **Assertion:** The SW Dev Programming Model for the Itanium relies too strongly on complex Programming Model (e.g. IPA, PFA, hand-tuning).
  - Hammer64 will not rely so much on these 'advanced' capabilities; however, they may be useful at times if used properly.

Fortran, C, C++
Scalar & OpenMP
Thread-Parallel
Compilers

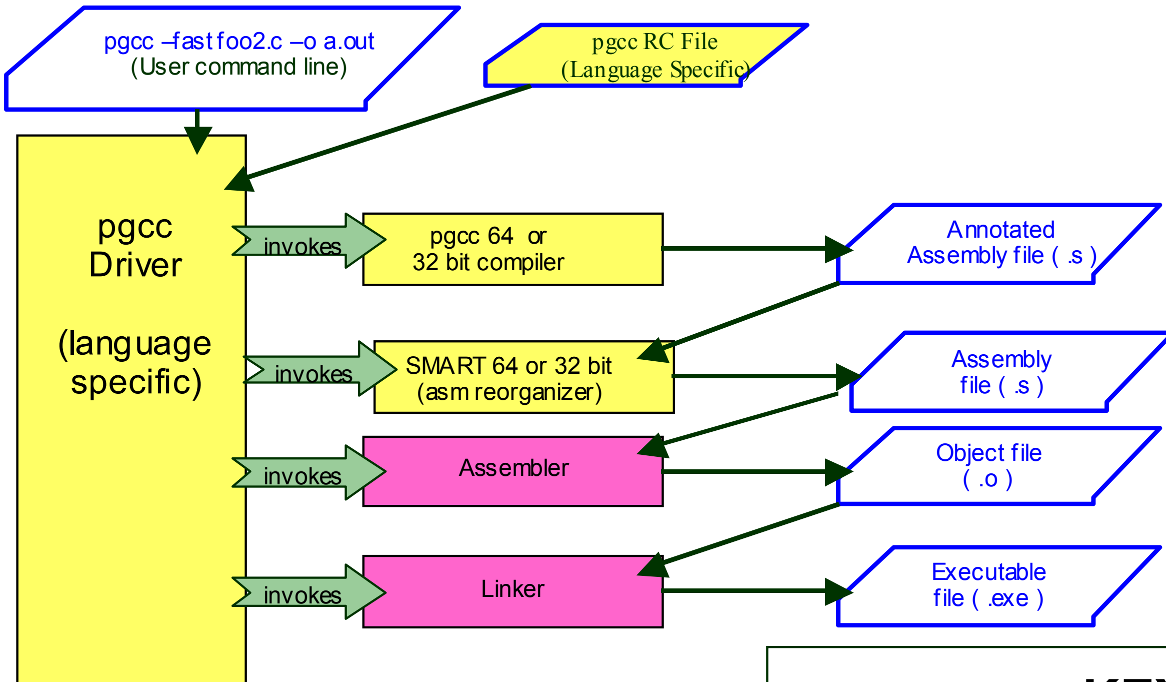
The PGI Compilers



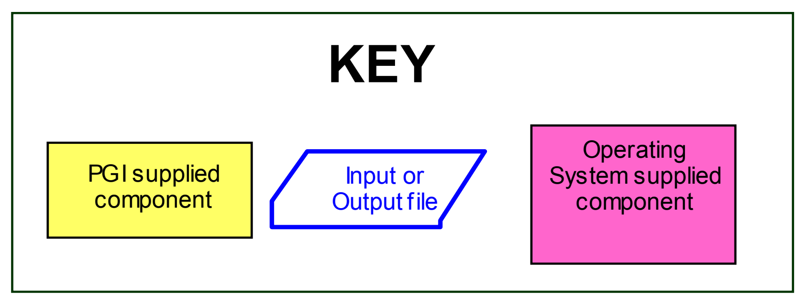
Hammer64 Compilers

- 64-bit addressing
- 64-bit integer registers & operations
  - No longer software-simulated
- SSE/SSE2 for floating-point operations
  - Helps with porting applications from non-x86 systems, IEEE
  - X86 FP will be used w/ Hammer32 when necessary (ABI, Performance)
- More General Purpose & SSE/SSE2 registers
- Better ABI (sanity ruled)
  - e.g. reduced call overhead due to arg passing in registers
- Fortran data-types unaffected
  - INTEGER (32-bit), INTEGER*8 (64-bit), F90 Kind parameters still the same, F90 REAL & DOUBLE PRECISION the same
- WIN64: long == 32-bits;   Linux64: long == 64 bits

Tool Chain Flow Block Diagram

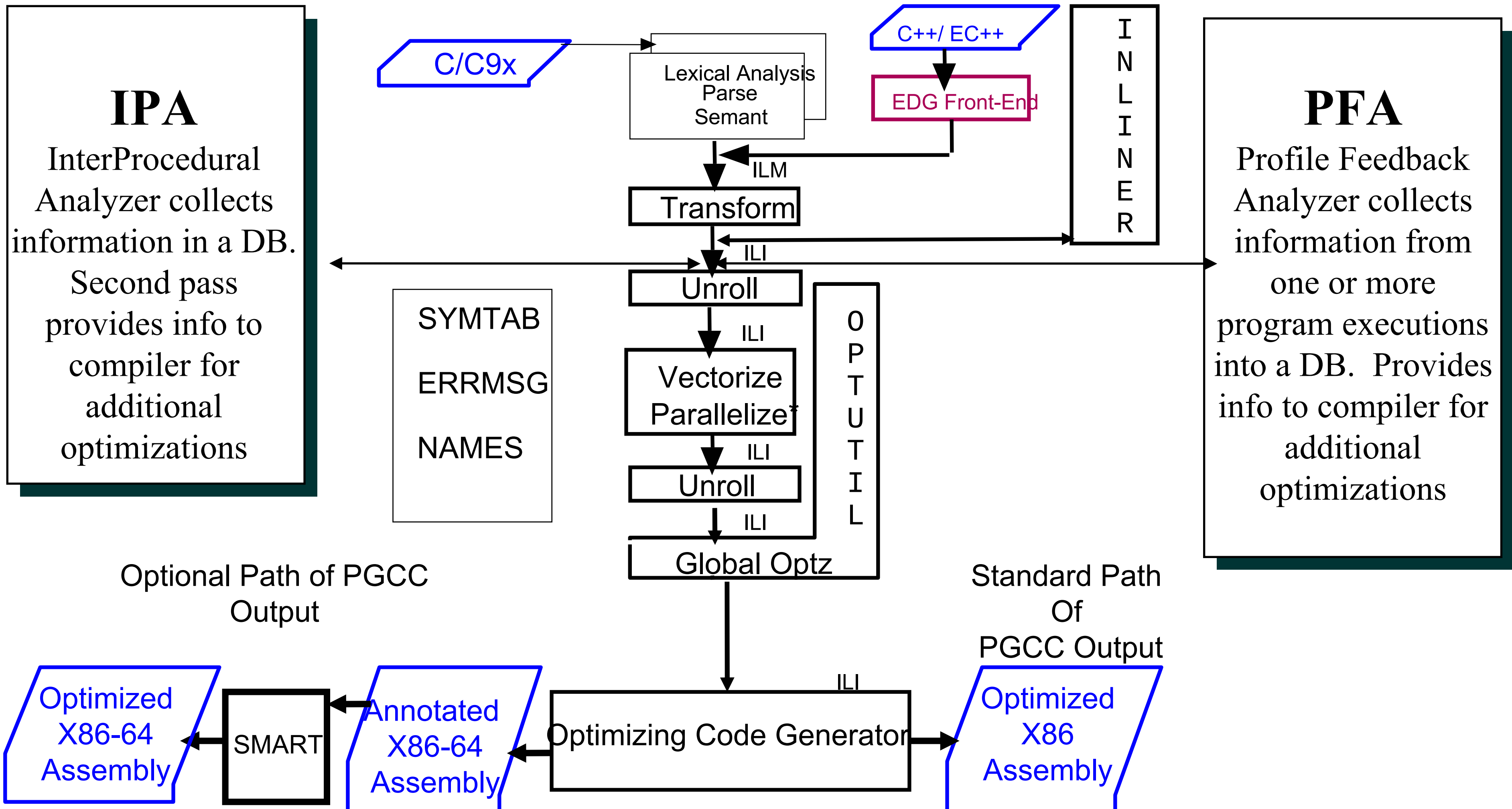


NOTE: If IPA or PFA is used, then a separate language specific IPA or PFA executable is invoked before the language specific compiler is invoked.

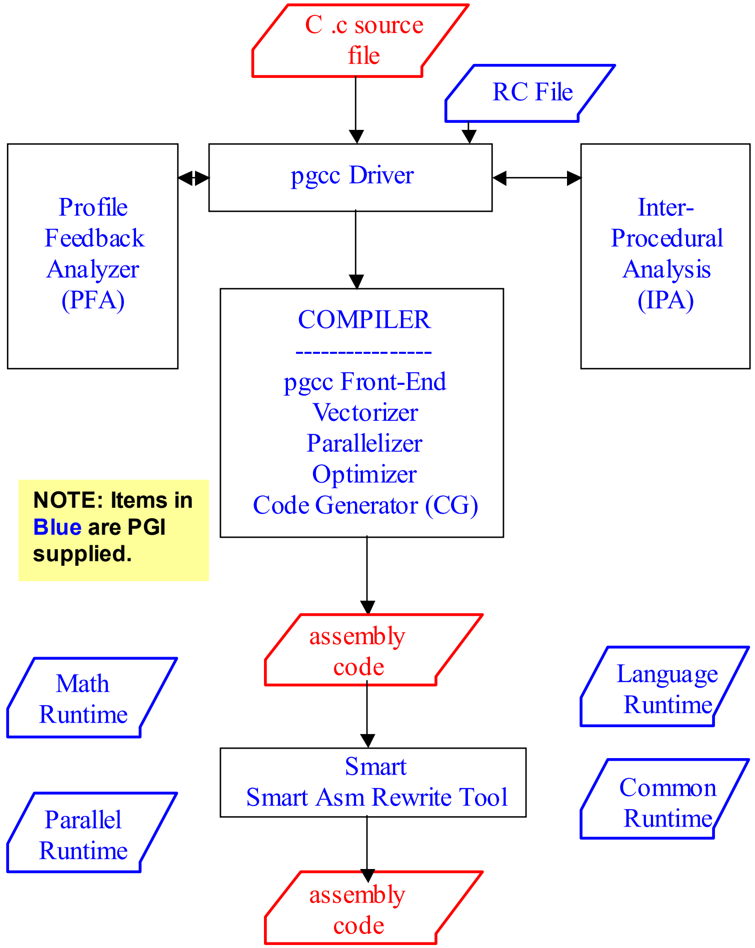


The Portland Group Compiler Technology

PGCC Block Diagram



PGCC Component Block Diagram



Fortran/C/C++ Key Features

- PGF90® Fortran 90 Compiler
  - Robust, commercially-supported, F90 w/ F95 features
- PGC++ C++ Compiler
  - EDG 2.40 C++ support for ANSI C++
  - EDG Microsoft Compatibility Switch
  - STLport v4.5 Standard Template Library
- PGCC ANSI C Compiler w/ K&R support
- OpenMP & Auto-parallel for SMP Servers – F77, F90, C, C++
- Hammer64 specific - More Registers, Scalar SSE (No x87 FP), SMART Asm Optimizer, Tuned Intrinsics, Better ABI
- Hammer: SMART Asm Optimizer, SSE/SSE2, x86 FP when needed on x86-32, x86-32 ABI, IPA, PFA, Cache optimizations (e.g. tiling), Cache-aware Vectorizer
- Switches: -fast -fastsse -Mvect=fast -Mnontemporal -help -Minfo=all
- Miscellaneous Fortran: byte-swapping I/O & Large File support
- GNU Interoperability On Linux - cross-link *gcc/g77* objects/libraries

Example Vectorize & SSE

Candidate loops are vectorizable and utilize SSE instructions:

```
% pgcc -fastsse -Mvect=sse -Minfo matmul32.c
main:
    29, Interchange produces reordered loop nest: 30, 29
        Loop unrolled 10 times
    30, Outer loop unroll (4 times) and jam
    34, Interchange produces reordered loop nest: 35, 34
        Loop unrolled 10 times
    35, Outer loop unroll (4 times) and jam
    46, Outer loop unroll (4 times) and jam
    47, Loop unrolled 10 times
    53, Loop unrolled 10 times
    58, Call to __pgi_adotp4 generated
Linking:
%
```

InterProcedural Analysis (IPA)

- **-Mipa=safe** when the whole call tree is not available to IPA, declares that unknown functions do not call back into the known functions, and do not change global variables
- **-Mipa=fast** common -Mipa options: -Mipa=const,globals,ptr,vestigial
- **-Mipa=const** propagates constants
- **-Mipa=ptr** pointer disambiguation
- **-Mipa=arg** removes arguments replaced by -Mipa=ptr,const
- **-Mipa=vestigial** eliminates functions that are not called
- **-Mipa=global** optimizes references to globals when not used in function call
- **-Mipa** equivalent to -Mipa=const

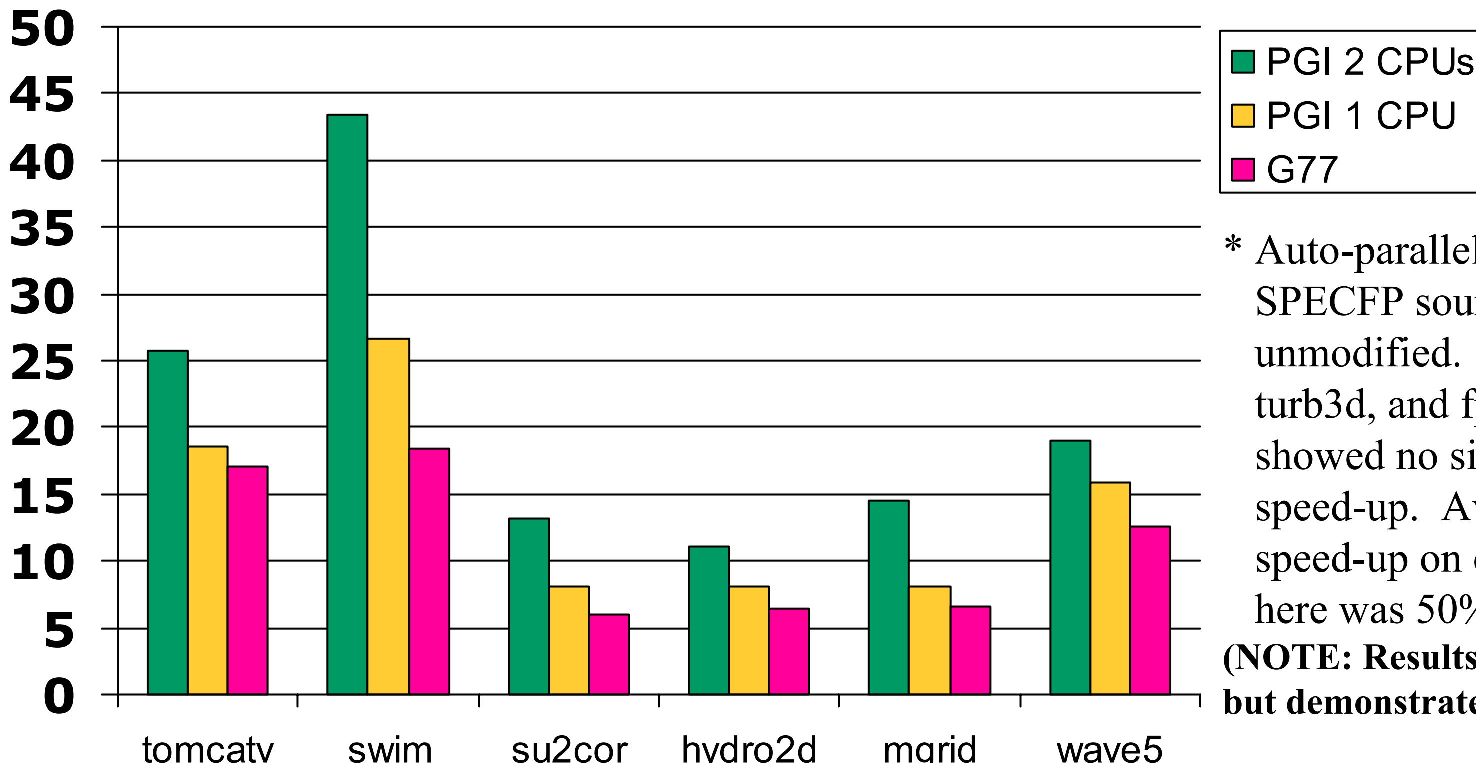
Parallel Programming Models

- MPI
  - *Low-level* library-based Fortran/C/C++ explicit parallel programming for *shared-memory*, *distributed-memory*, and *hybrid-memory* servers and clusters
- Automatic SMP
  - Compiler switches that direct the Fortran/C/C++ compiler to automatically detect loop-level parallel opportunities and create threaded parallel code. (SMP *shared-memory servers only*)
- OpenMP
  - *High-level* directive-based Fortran/C/C++ thread-parallel programming (SMP *shared-memory servers only)*
- HPF
  - *High-level* directive-based Fortran parallel programming for *shared-memory*, *distributed-memory*, and *hybrid-memory* servers & clusters

SPECFP_base95 w/ Auto-parallelization

PGI Options: **-fast –Mconcur –Minline**

G77/EGCS Options: **-O3 -fomit-frame-pointer -funroll-loops -fstrength-reduce -fexpensive-optimizations -ffast-math -malign-double**



* Auto-parallel only. SPECFP source completely unmodified. The applu, apsi, turb3d, and fpppp bmks showed no significant speed-up. Average parallel speed-up on codes shown here was 50% on 2 CPUs.

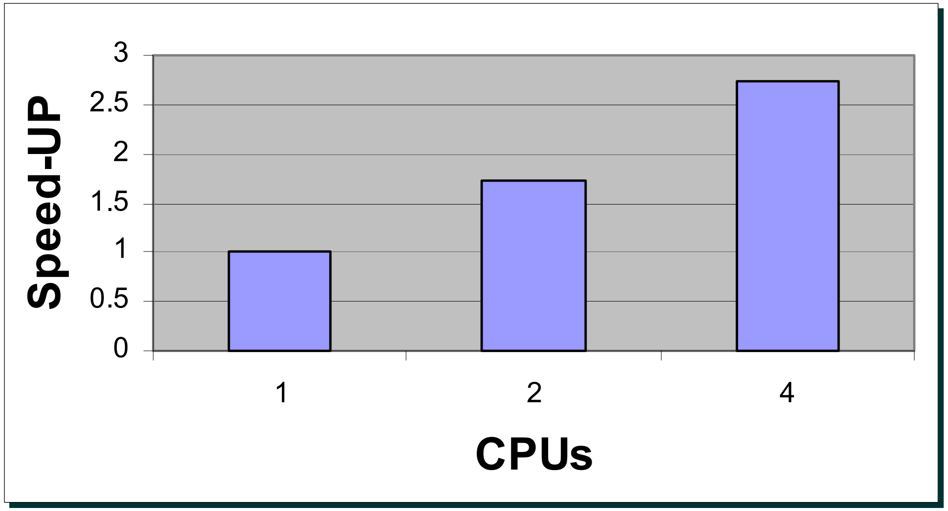
(NOTE: Results are quite dated but demonstrate auto-parallel.)

Example OpenMP Code

AMD DEVELOPER symposium 2002

```
    for (l=1; l<=NTIMES; l++) {
#pragma omp parallel private (j,i,ii,k)
      {
#pragma omp for
      for (j=0; j<p; j++) {
        for (i=0; i<m; i++) {
          c[j][i] = 0.0;
        }
      }
      for (i=0; i<m; i++) {
#pragma omp for
        for (ii=0; ii<n; ii++) {
          arow[ii] = a[ii][i];
        }
#pragma omp for
        for (j=0; j<p; j++) {
          for (k=0; k<n; k++) {
            c[j][i] = c[j][i] + arow[k] * b[j][k];
   } }  } }
    (void) dummy(c);
    }
```

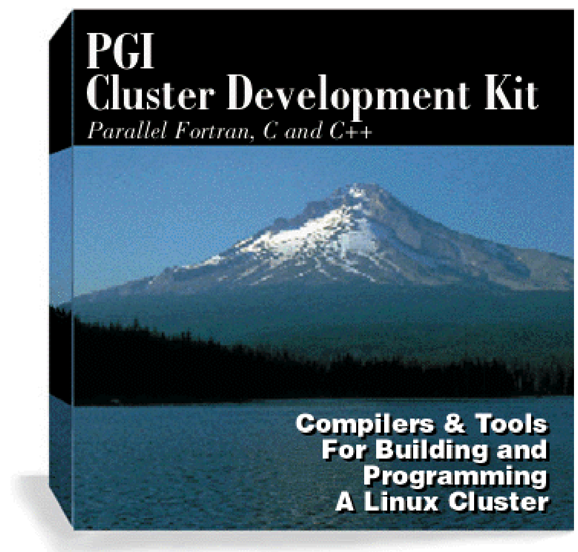
LS-DYNA under OpenMP



111029 Elements
1 CPU Time = 119955 Seconds
2 CPU Time = 69113 Seconds
4 CPU Time = 43864 Seconds
Quad 500 Mhz Xeon

- LSDYNA = Over 750,000 lines of Fortran and C source code
- 1.74X Speedup on 2 CPUs, 2.74X Speedup on 4 CPUs
- Parallelized using *PGI Workstation* Fortran and OpenMP

Debuggers
Profilers
Tools



OpenMP & MPI Cluster PGDBG®

- **Debugs Cluster-level MPI + OpenMP Thread-level Programs**

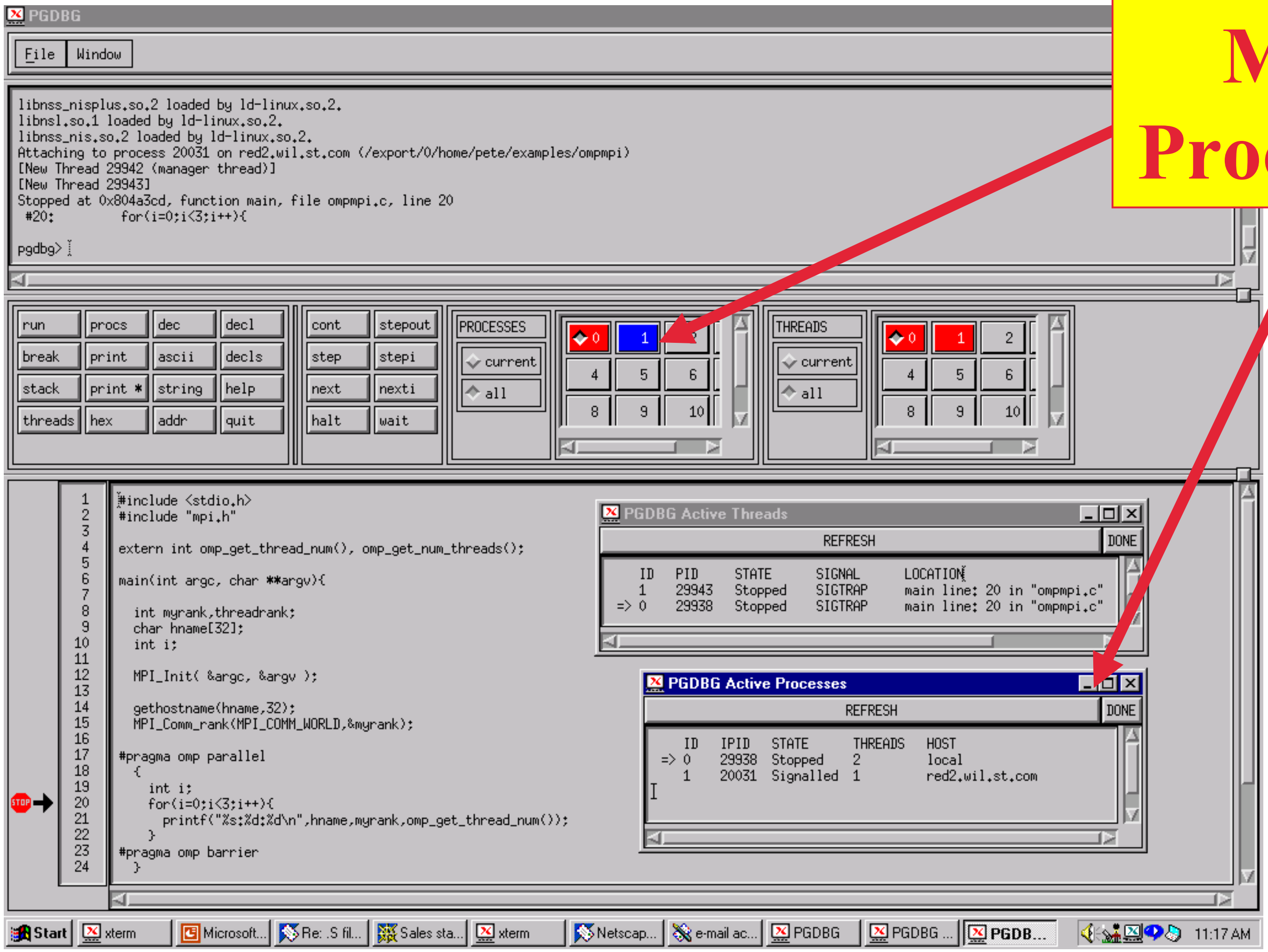
- **For each or all Processes AND for each or all Threads:**
  - Step, Break, Examine variables, Continue, …

- **Shared-Memory Debugging**
  - OpenMP or Automatic Thread-level parallelism
  - For each OR for all Threads – Step, Break, Halt, Windows, Continue,…

- **Distributed-Memory MPI Debugging**
  - Debug MPI Processes
  - For each OR for all Processes – Step, Break, Halt, Windows, Continue,…

- **Hybrid-Memory MPI + OpenMP Debugging**
  - Multiple Nodes where each node has more than one SMP CPU
  - Control each process & control each thread separately or as groups.

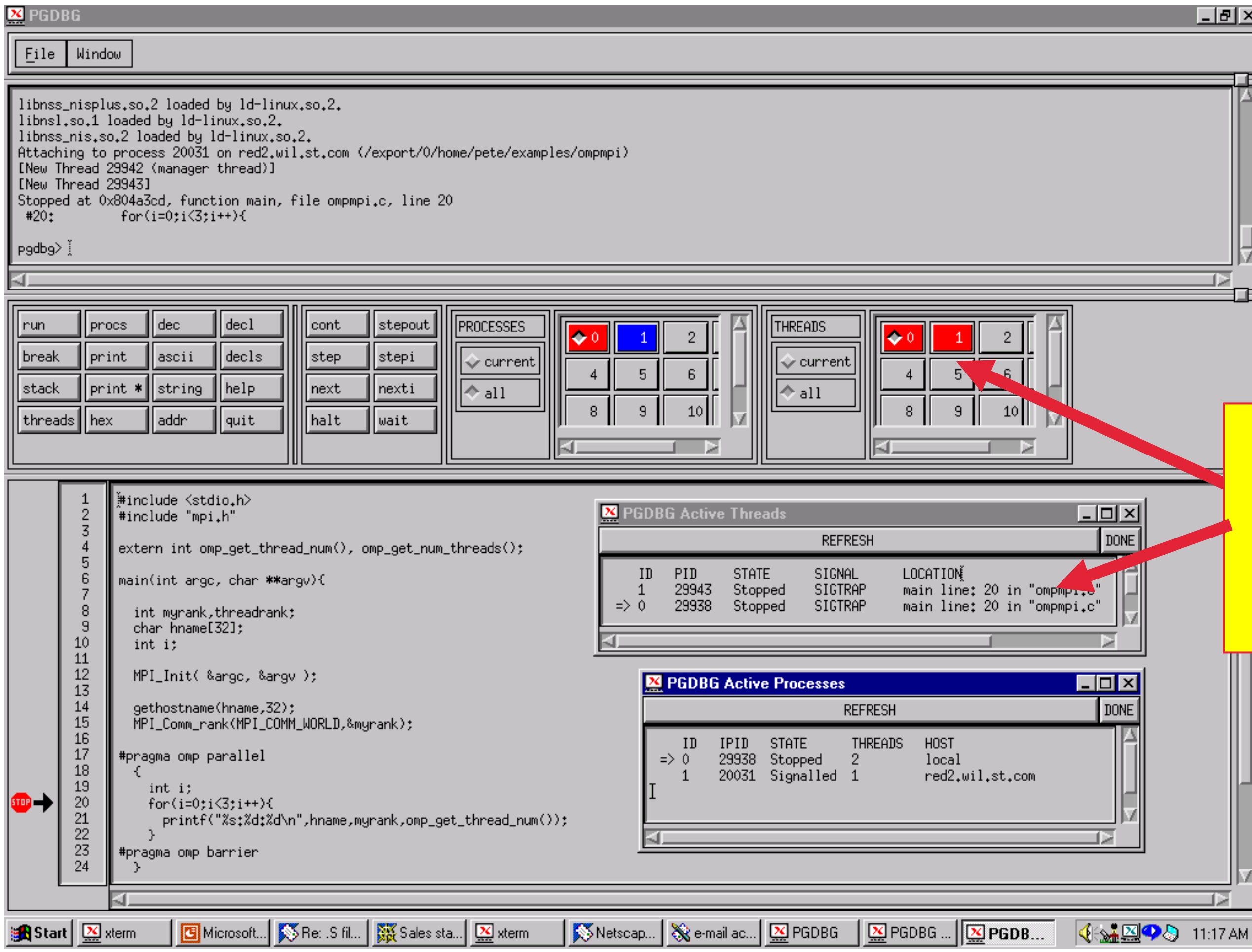
MPI Process Control



Control MPI Processes

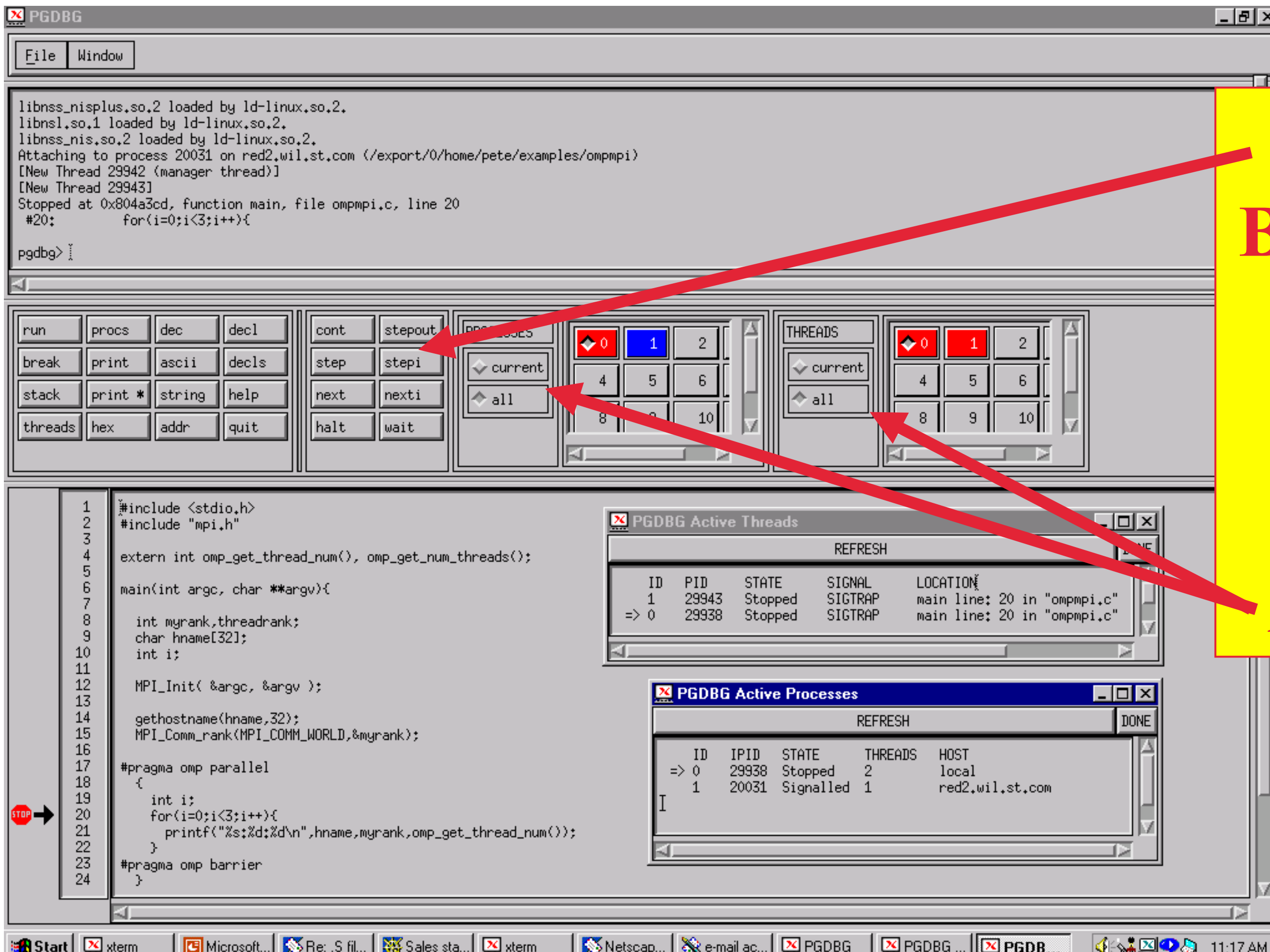
The Portland Group Compiler Technology

Thread Control



Control Threads

Process + Thread Controls



Step, Next, Break, Halt, Wait or Continue on Threads or Processes

All or Individually

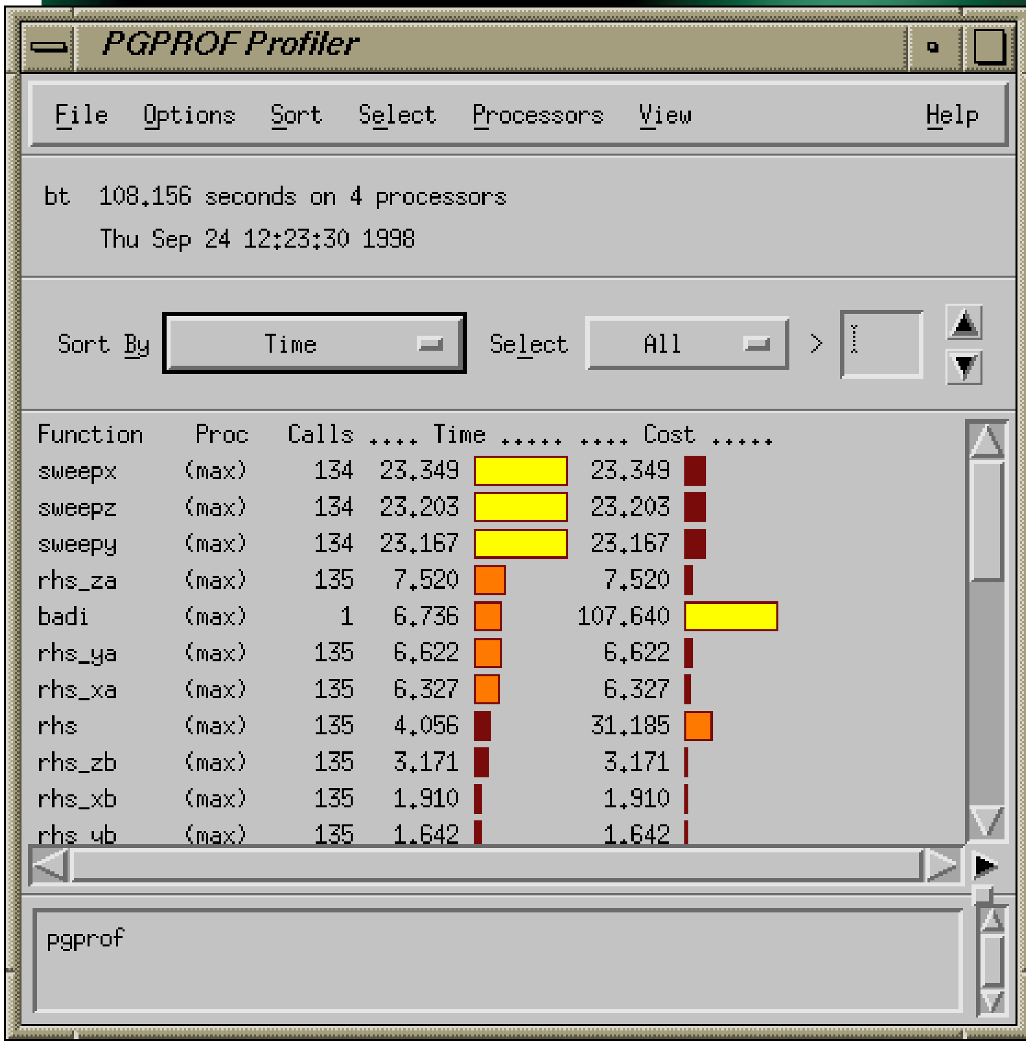
Cluster MPI + OpenMP PGPROF

- **Profile Cluster-level MPI + OpenMP Thread-level Programs**
- **For each or all Processes AND for each or all Threads:**
  - –Examine Maximum, Minimum or Average Time spent in each or all.
  - –Examine each process or thread individually
- **Shared-Memory Post-Mortem Profiling**
  - –OpenMP or Automatic Thread-level parallelism
  - –Examine Counts, Time or Cost on a Function Level or Statement Level
- **Distributed-Memory MPI Post-Mortem Profiling**
  - –Profile MPI Processes and HPF Programs
  - –Examine Counts, Time or Cost on Function Level or Statement Level
  - –Examine Messages Sent and Messages Received
- **Hybrid-Memory MPI + OpenMP Profiling**

The Portland Group Compiler Technology

Cluster PGPROF Profiler

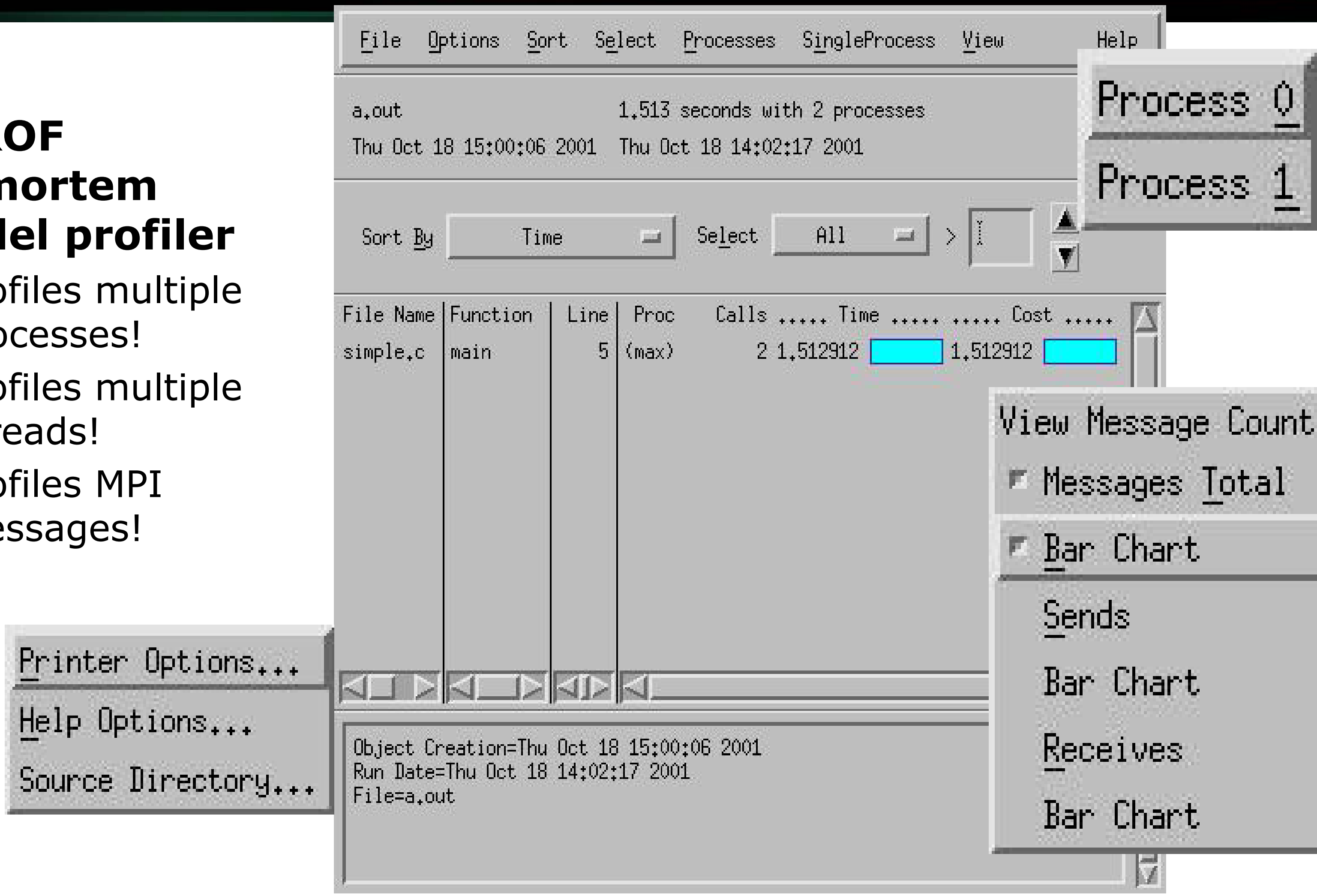
AMD DEVELOPER symposium 2002



- *Cluster Profiling*
  - *Threads, Processes, Messages*
  - *OpenMP, MPI, HPF, Scalar*
  - *MPI Profiling*
    - *Send/Recv/Counts*
- *Post-Mortem*
- *Multi-Window*
- *Function Level*
- *Source Line Level*
- *C, some C++ &F90*
- *GUI & non-GUI*

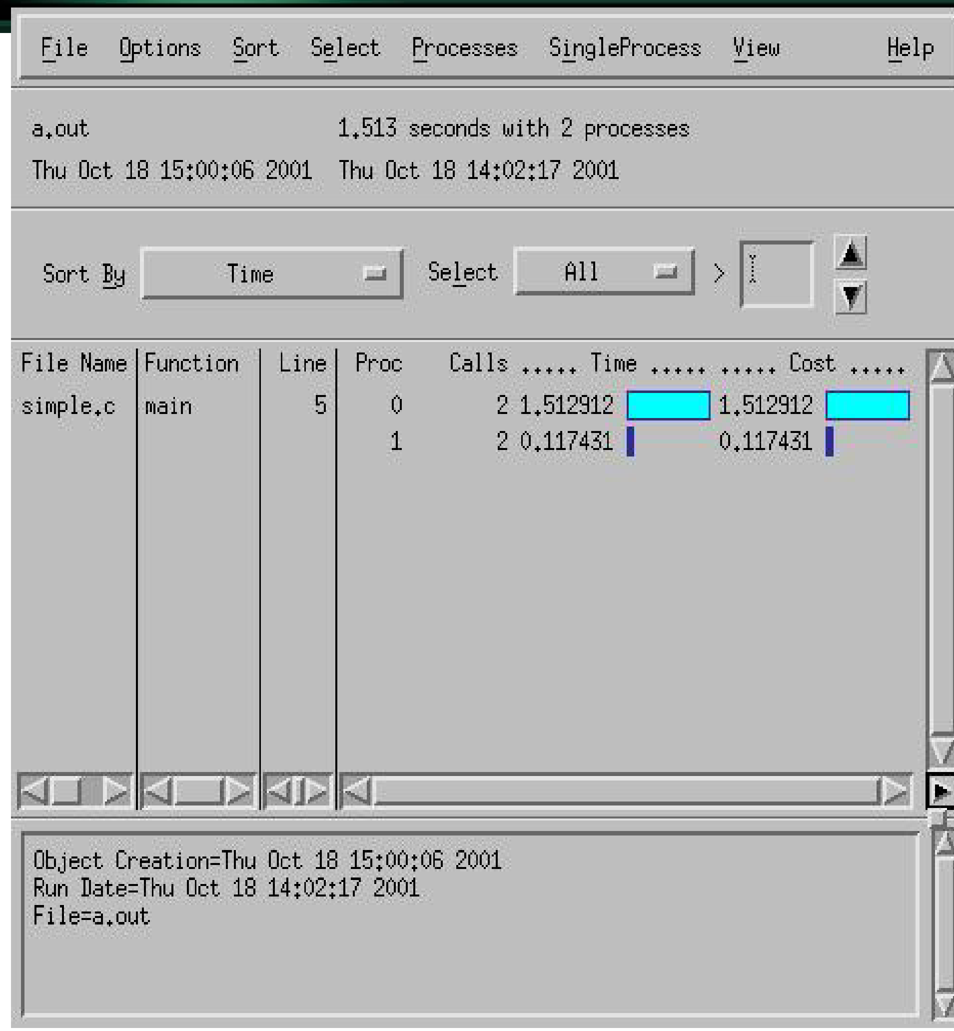
Parallel PGPROF

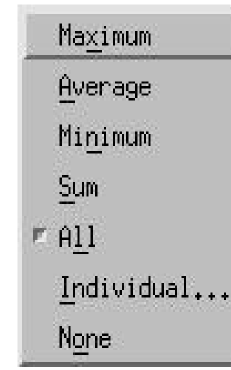
- **PGPROF postmortem parallel profiler**
  - Profiles multiple processes!
  - Profiles multiple threads!
  - Profiles MPI messages!

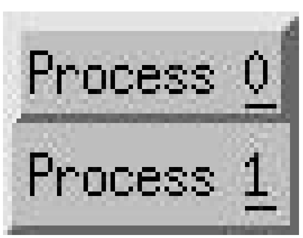


Function-level Process Profile

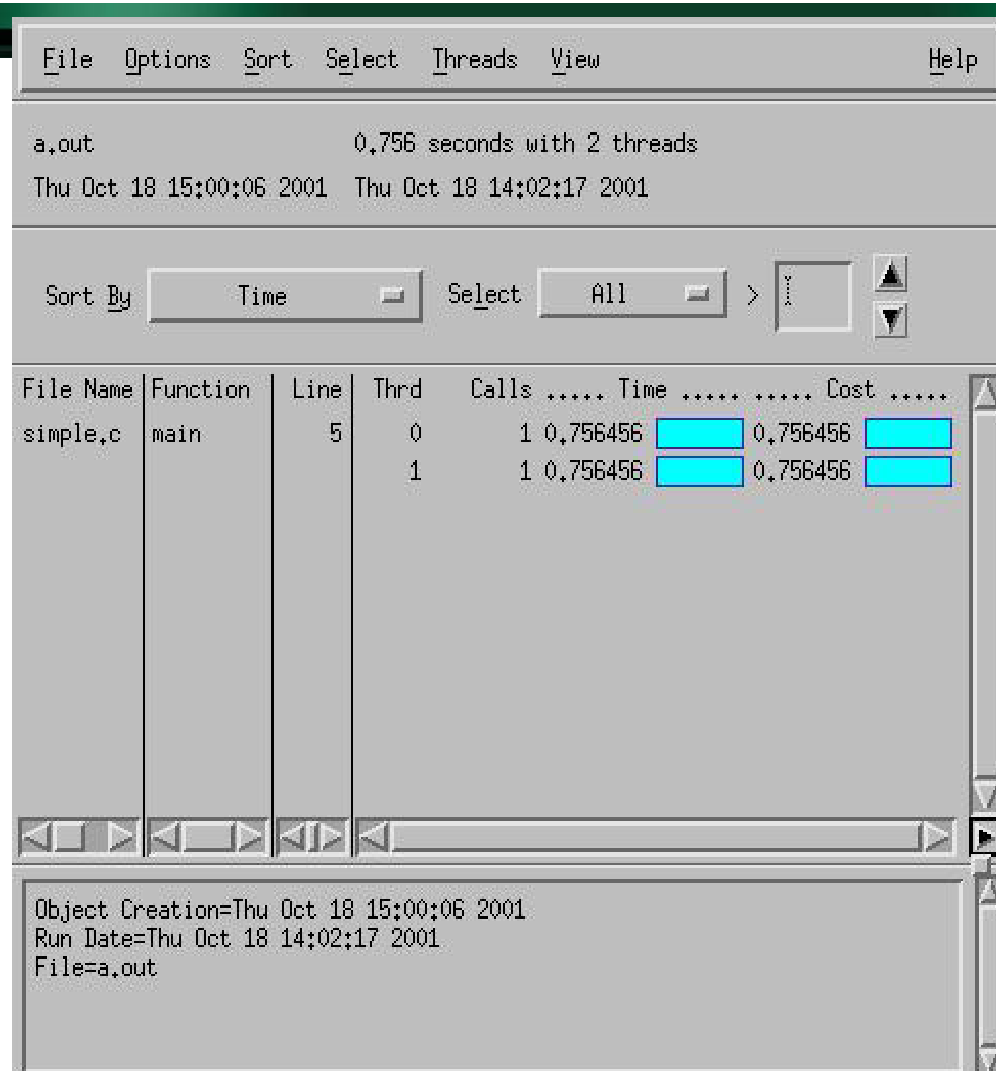
- For **All** processes, observe:
  - Number of calls to each function
  - Time in each call
  - Total cost for each call
- OR select the processes individually







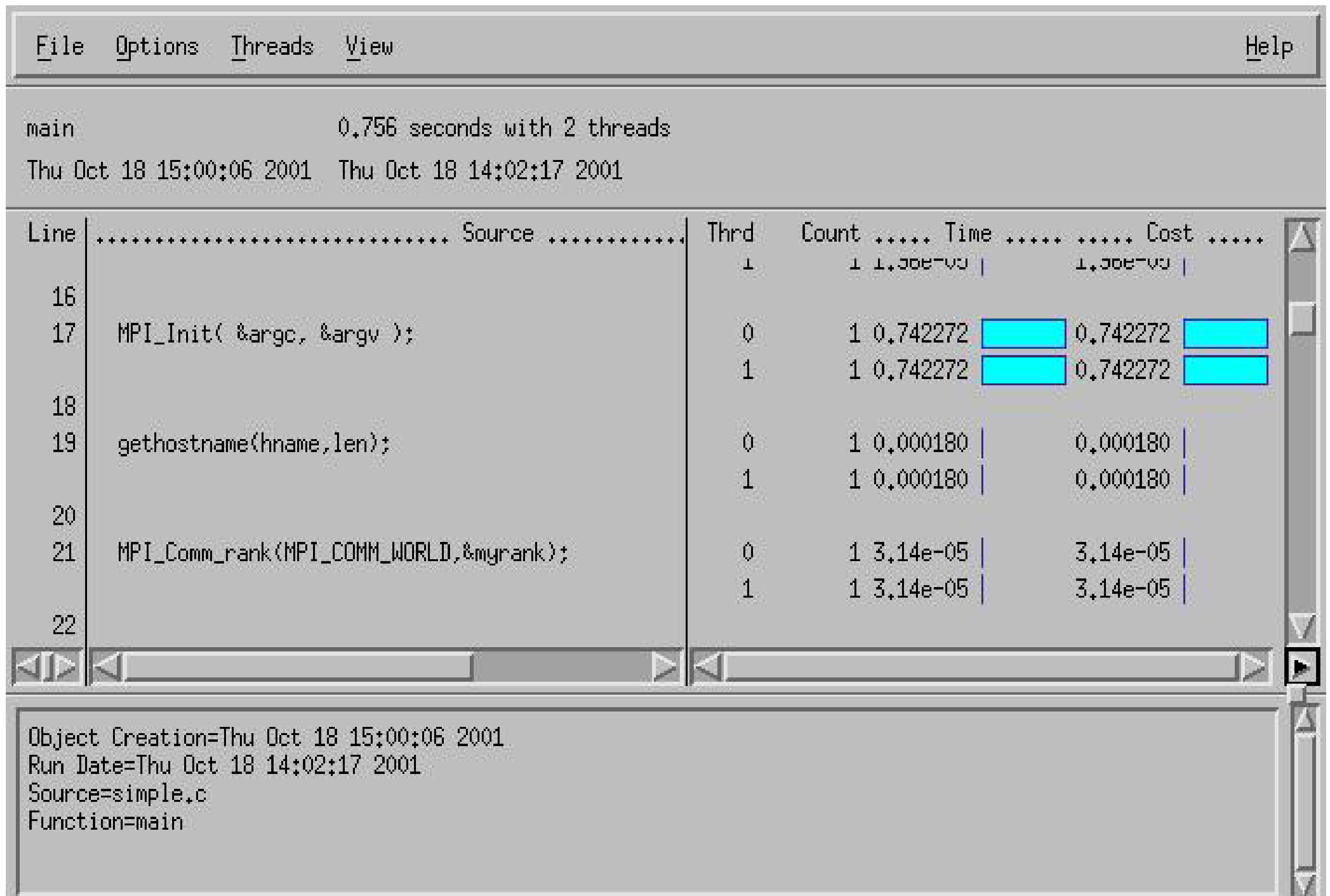
Function-level Thread Profile



- For each Thread (**Thrd**), observe:
  - Number of **Calls** to each function
  - **Time** in each call
  - Total **Cost** for each call
- **Time**
  - The amount of time spent within the function.
- **Cost**
  - The amount of time spent to execute the function and all children from that function.

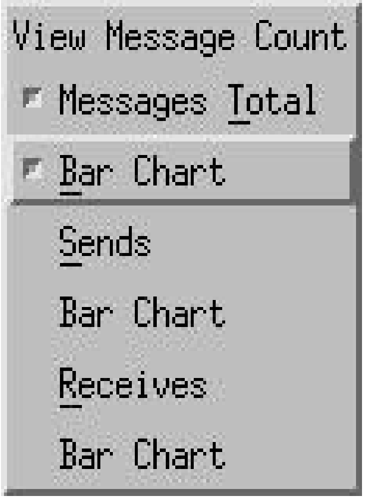
Line-level Thread Profile

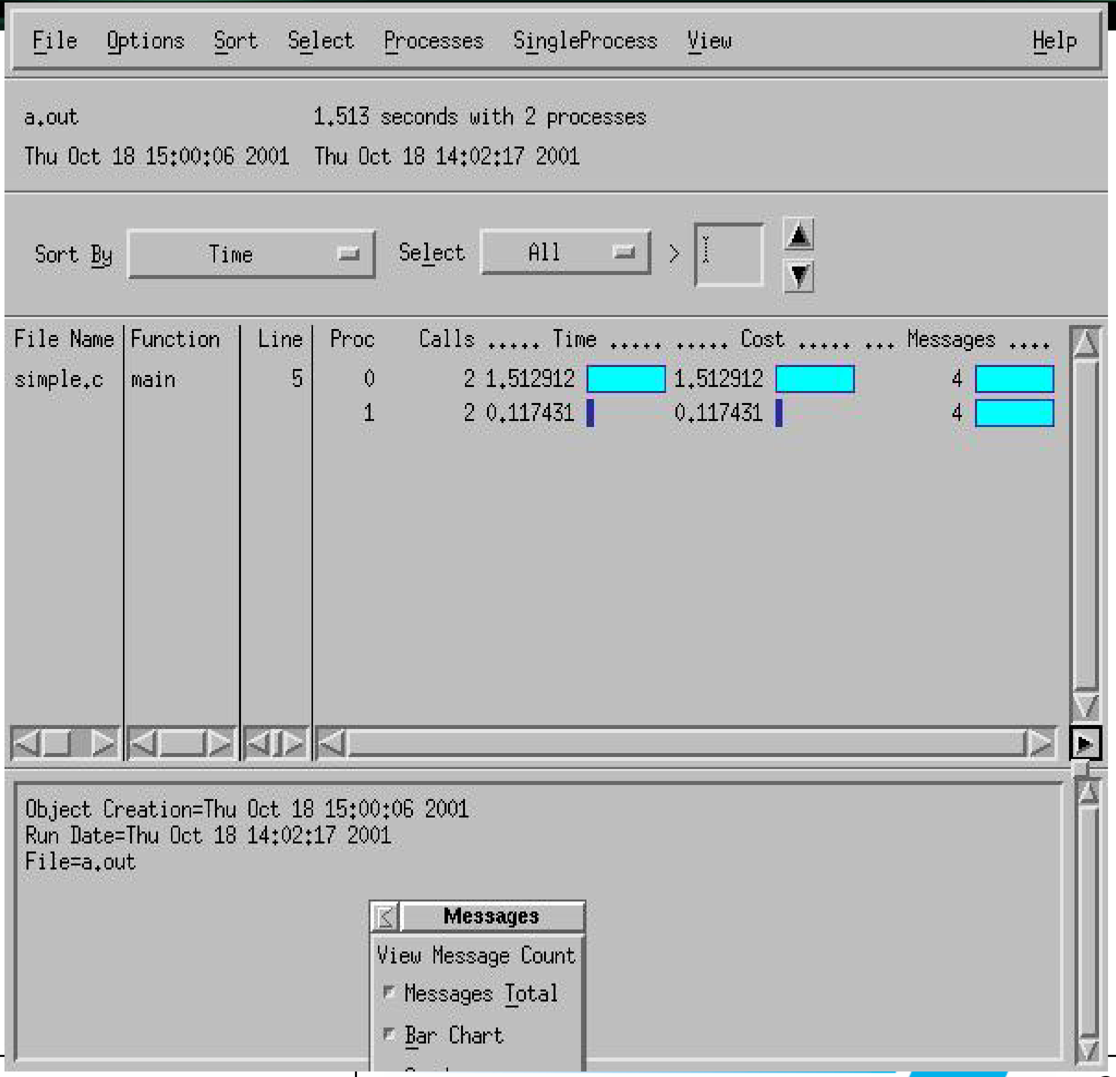
- **Within a function, at a line-level, observe for each thread:**
  - Number of times each line was executed
  - Time for each line
  - Total cost for each line



Function-level MPI Profile

- For each function and for each process, you can observe the number of MPI messages
- Determine the number of Sends
- Or of Receives





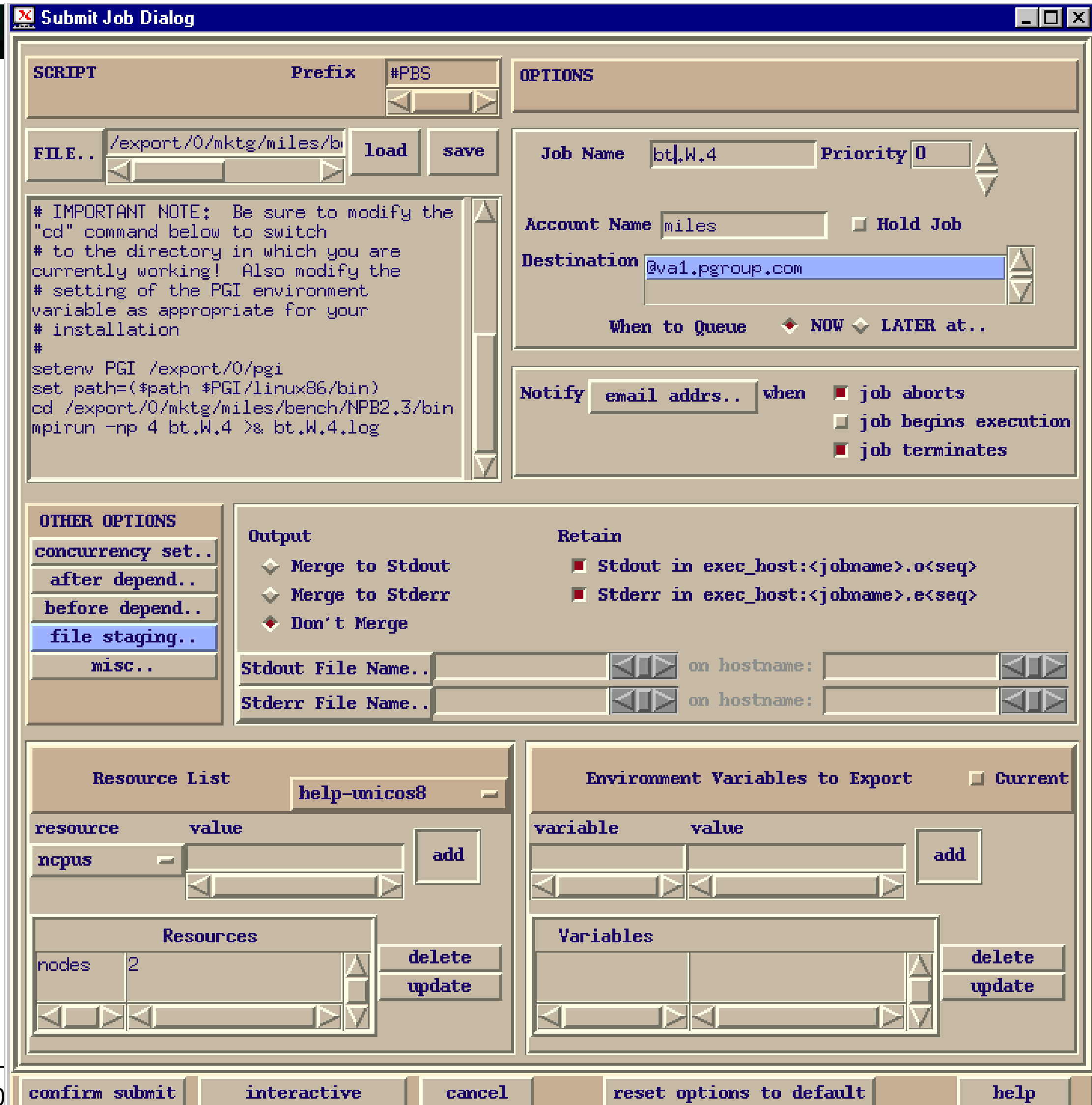
CDK Clustering Software

- ***MPI-CH*** - Pre-configured libraries and utilities for ethernet-based IA32/Linux clusters (http://www-unix.mcs.anl.gov/mpi)
- ***PBS*** – Portable Batch System batch-queueing from NASA Ames and MRJ Technologies (http://pbs.mrj.com)
- ***ScaLAPACK*** - Pre-compiled distributed-memory parallel Math Library
- ***Training*** – Tutorials (OSC), exercises, examples and benchmarks for MPI, OpenMP and HPF programming

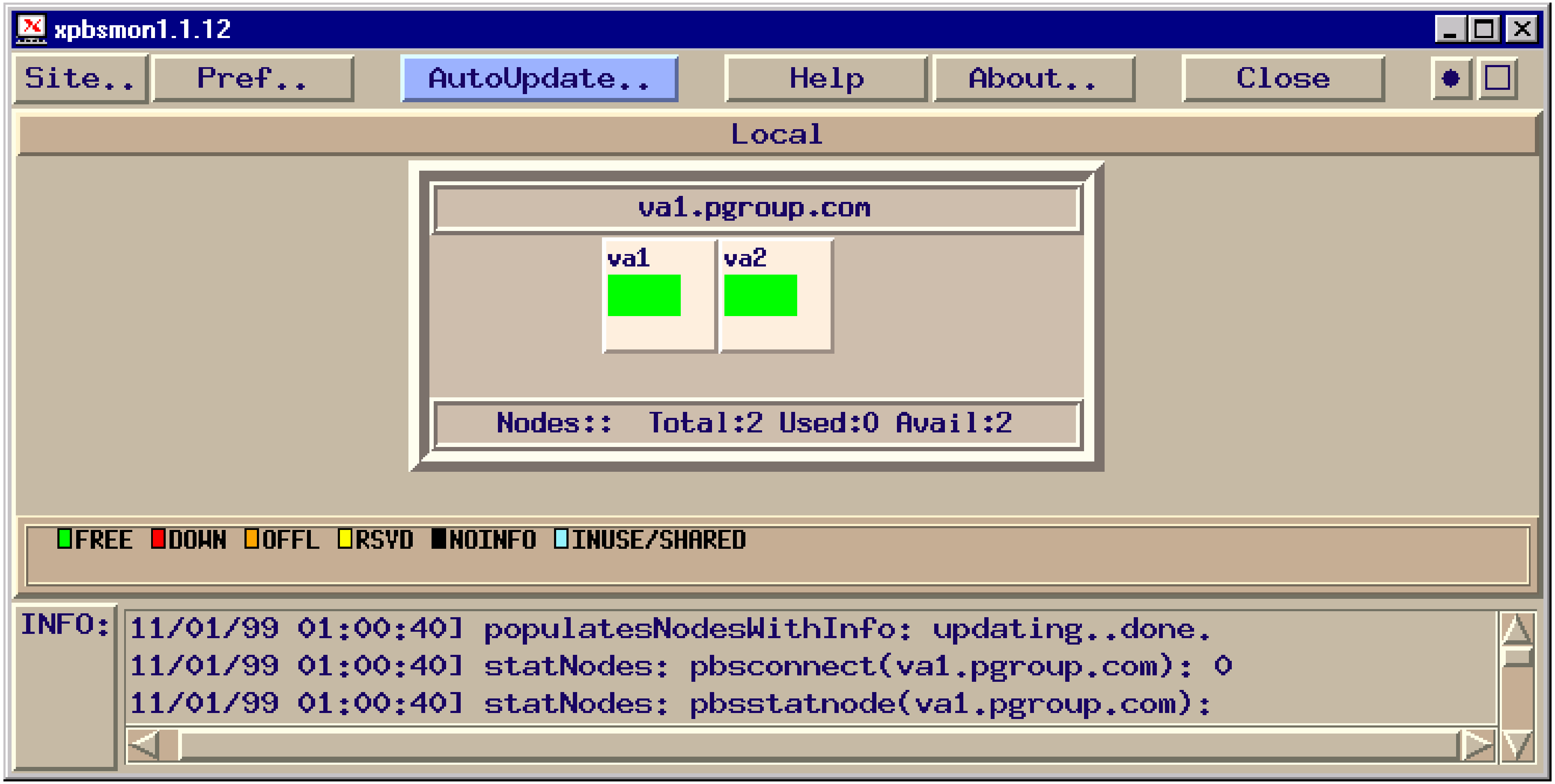
PBS Batch Queuing System

- Configured by default as a single space-shared queue – PBS is very configurable! Space-shared and time-shared queuing options available, custom scheduling available
- Option of having a dedicated Front-end Node or using one of the Compute Nodes to submit/monitor jobs
- Supports both interactive and batch jobs
- Complies with POSIX 1003.2d standard for shells, utilities, and batch environments
- Available on many platforms besides Linux – PBS is gaining popularity as the de facto standard open source Queuing utility
- Commercial support available from MRJ Technologies

XPBS – Job Submission GUI



XPBSMON – Job Monitoring GUI



Schedule

Schedule

- Linux64
  - Beta versions of C, Fortran, C++ compilers prior to Hammer launch
  - Commercial Compiler Release in 1H03
  - Commercial Parallel Debugger/Profiler Release in Q4 03
  - Current Effort is On track
- Windows64 Goals
  - Beta versions of C, Fortran, C++ compilers prior to Win64 commercial release
  - Commercial Compilers in conjunction with Win64 commercial release
  - Commercial Compilers in MingW Win32 environment
  - Commercial Parallel Debugger/Profiler about 3-6 months after compilers

Trademark Attribution